VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2020 Patrik Szewczyk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Patrik Szewczyk**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ATLAS consulting spol. s r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Marek Běhálek, Ph.D.**

Konzultant bakalářské práce: Mgr. Tomáš Řehák

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2016 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 27. dubna 2020

ATLAS consulting spol. s r.o. (14)
Výstavní 292/13
702 00 Ostrava, Moravská Ostrava
IČO 46578706, DIČ CZ46578706

Rád bych poděkoval všem, kteří mi s prací pomohli, všem kolegům a kolegyním ze společnosti ATLAS Consulting s.r.o. a vedoucímu mé odborné práce, panu Ing. Markovi Běhálkovi, Ph.D., protože bez nich by tato práce nevznikla.

Abstrakt

Tato bakalářská práce se zabývá průběhem absolvování individuální odborné praxe ve společnosti Atlas consulting s r.o. [1], v hlavní části této bakalářské práce je rozepsána především náplň práce a řešené úkoly na komplexní webové aplikaci Codexis [2], na jehož vývoji jsem měl možnost se podílet. V závěru práce jsou shrnuty teoretické i praktické znalosti a dosažené výsledky, jenž byly získány za dobu studia a mohly být uplatněny během této individální praxe.

Klíčová slova: JavaScript, TypeScript, React

Abstract

This bachelor thesis describes the course of completion of individual professional practice in the company Atlas consulting s r.o. [1], in the main part of this bachelor thesis is described the content of the work and the tasks solved on a complex web application Codexis [2], in whose development I had the opportunity to participate. The conclusion summarizes the theoretical and practical knowledge and results obtained during the study period and could be applied during this individual practice.

Keywords: JavaScript, TypeScript, React

Obsah

Seznam použitých zkratek a symbolů

XML	–	Extensible Markup Language
HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
JSX	–	JavaScript XMXL
CSS-in-JS	–	Cascading Style Sheets in JavaScript
DOM	–	Document Object Model
UX	–	User Experience
UI	–	User Interface
API	–	Application Programming Interface

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Jsem opravdu rád, že jsem si mohl zvolit téma bakalářské práce formou absolvování bakalářské praxe, jelikož z mého pohledu je opravdu důležité najít průnik mezi teoretickými znalostmi, které se studenti učí nejen ve škole, a mezi praktickými zkušenostmi, kterých člověk nabývá právě postupnou odbornou praxí v daném oboru. Lidé, kteří najdou tento průnik a pořád se v obou směrech zdokonalují, jsou podle mě opravdoví odborníci v daném oboru, již jsou k nezaplacení.

V následujících kapitolách představím společnost ATLAS Consulting s.r.o. [1] a software, který tato společnost vyvíjí a poskytuje, následně ve zkratce popíši technologie, použité na vývoji software **Codexis** [2], jenž je hlavním projektem, kterým se budu v bakalářské práci zabývat.

Primární část této práce tvoří rozbor a řešení jednotlivých úkolů, se kterými jsem se během odborné bakalářské praxe potýkal. Od vytváření jednoduchých komponent do abstraktní knihovny **React Atlantic**, přes opravy, mazání a přepisování původních zdrojových kódu v aplikaci Codexis nebo například řešení animací, až po komplexní asynchronní a částečné vykreslování gigantických HTML dokumentů do prohlížeče. V této části se také nachází kapitola o nejnovějším software, jehož vývoj probíhal plně distančně a velmi rychle.

V závěru práce proběhne shrnutí a zhodnocení právě teoretických a praktických znalostí, které byly získány v průběhu studia a znalosti, kterých jsem dosáhnul během této odborné bakalářské praxe.

2 Popis firmy ATLAS Consulting s.r.o.

ATLAS consulting spol. s r.o. člen skupiny ATLAS GROUP je ryze českou softwarovou společností, která se již od roku 1992 věnuje vývoji právních a manažerských informačních systémů a aplikací. Společnost ATLAS software a.s. člen skupiny ATLAS GROUP byla založena v roce 2001 a od doby svého vzniku zajišťuje výhradní obchodní zastoupení společnosti ATLAS consulting spol. s r.o. Prodej, distribuci, školení a poradenství v oblasti software a hardware zajišťuje síť obchodních zástupců a školitelů. Péči o zákazníky a technickou podporu poskytuje společnost prostřednictvím vlastního klientského centra, zákaznické podpory a servisních pracovníků s celorepublikovou působností.[1]

2.1 Codexis

Nejvýznamnějším produktem je bezesporu právní informační systém CODEXIS [2], který je nejrozšířenějším a nejkomplexnějším právním informačním systémem v České republice. Je zcela jedinečný díky svému zaměření, úplnosti rozsahu, originálnímu způsobu zpracování, pestré řadě funkcí a mnohým dalším užitečným vlastnostem. Tento software je určen především pro právníky a zástupce právnických profesí, finanční manažery, ekonomy, účetní, úředníky, manažery a specialisty, kteří k výkonu své práce potřebují snadný přístup k právním předpisům, zákonům a informacím k určitému tématu. Codexis obsahuje veškeré zákony spadající do legislativy České republiky a legislativy Evropské unie, všechny zákony jsou navíc uvedeny v aktuálním, historickém i budoucím znění, díky čemuž mají uživatelé jistotu, že je daný dokument kompletní. CODEXIS svým uživatelům nabízí také možnost zasílání upozorňování na změny u sledovaných předpisů. Software obsahuje také judikaturu České republiky a judikatury Evropské unie.

2.2 Projekt X

Jedná se o nejnovější produkt společnosti Atlas Consulting s.r.o.. Tento projekt je možné nazvat jinými slovy jako právní poradna online. Hlavním motivem k vytvoření tohoto projektu bylo spojit advokátní kanceláře s širokou veřejností. Je určen pro lidi, kteří mají různé právní problémy a potřebují nezávisle kontaktovat vícero advokátních kanceláří zároveň. Projekt byl v první iteraci prezentovatelný za pouhý jeden týden.

3 Klíčové technologie

V následujících bodech budou shrnuty zásadní technologie, jež byly využity při práci na projektech a řešených úkolech.

JavaScript

JavaScript [3] je interpretovaný, objektově orientovaný jazyk, především známý jako skriptovací jazyk webových stránek, ale používá se také v mnoha prostředích bez webového prohlížeče. Je to skriptovací jazyk založený na prototypech, je dynamický a podporuje objektové, imperativní i funkcionální programovací paradigma.

TypeScript

TypeScript [4] je open-source programovací jazyk. Jedná se o nadstavbu nad jazykem JavaScript, která jej rozšiřuje o statické typování a další atributy, které známe z objektově orientovaného programování jako jsou třídy, moduly a další. Samotný kód psaný v jazyce TypeScript se kompiluje do jazyka JavaScript. Jelikož je tento jazyk nadstavbou nad JavaScriptem, je každý JavaScript kód automaticky validním TypeScript kódem.

React

React je open-source knihovna pro JavaScript, která se zabývá tvorbou uživatelského rozhraní.

1. **Komponenty a jejich vlastnosti:** Komponenty umožňují rozdělit UI na nezávislé, opakovaně použitelné kusy a přemýšlet o každém kuse izolovaně. Koncepčně jsou komponenty jako funkce v JavaScriptu. Přijímají libovolné vstupy, nazývané „vlastnosti“, které nelze měnit uvnitř komponenty. Vracejí React objekty, které popisují, co by se mělo na obrazovce objevit.[5]

2. **Stav a životní cyklus komponent:** Stav komponenty [6] je objekt, který obsahuje libovolné informace, které se mohou během životního cyklu komponenty měnit. React komponenty dodržují takzvaný životní cyklus, jsou vytvářeny (vytvořeny v DOM), aktualizovány a mohou být také zničeny (odebrány z DOM). V rámci životního cyklu komponenty existují v různých fázích, každá z těchto fází má své vlastní metody a vlastnosti životního cyklu.

3. **React Hooks:** Hooky [7] jsou funkce, které umožňují připojit ke komponentě specifickou funkcionalitu. Dokáží reagovat na stav a životní cyklus komponent. Hooky fungují pouze ve funkcích, nikoli ve třídách - umožňují používat React bez tříd a tímto zjednodušují zdrojový kód, díky čemuž je možno psát zdrojový kód plně podle funkcionálního programovacího paradigma.

4. **Context:** V typické React aplikaci jsou data předávána shora dolů (z rodiče na dítě) prostřednictvím vlastností, ale to může být těžkopádné pro určité typy vlastností (například téma uživatelského rozhraní), které jsou vyžadovány mnoha komponentami v aplikaci. Kontext poskytuje způsob, jak sdílet hodnoty jako tyto mezi komponentami, aniž by bylo nutné explicitně předávat vlastnosti přes každou úroveň stromu. [8]

GraphQL

GraphQL [9] je moderní, dotazovací jazyk pro API. Schéma tohoto API nabízí kompletní a srozumitelný popis dat, které daný server poskytuje, což znamená, že klient má možnost dotazovat přesně o ta data, která potřebuje.

Apollo

Apollo [10] je platforma, postavená na základním open source klientovi a serveru GraphQL, poskytuje vývojářské nástroje, služby pro urychlení vývoje a zabezpečení infrastruktury.

Styled Components

Knihovna Styled Components využívá principy a implementace kaskádových stylů uvnitř JavaScriptového kódu (CSS-in-JS), díky těmto principům umožňují komponenty plně zapouzdřit, izolovat nebo například dědit kaskádové styly z jiných stylovaných komponent.

React Virtualized

React Virtualized je komplexní knihovna, která řeší vykreslování komponent, které obsahují velký datový objem, jenž je potřeba zobrazit v prohlížeči. Knihovna na základě specifikovaných rozměrů a pozice vykresluje do DOM jen konkrétní interval těchto dat a zbytek drží v paměti prohlížeče.

React Spring

React Spring [11] je knihovna, která se specializuje na animace. Tyto animace jsou založeny na fyzice, která by měla pokrýt většinu animačních potřeb souvisejících s uživatelským rozhraním.

i18next

I18next [12] je internacionalizační nástroj napsaný v JavaScriptu. i18next jde nad rámec pouhého poskytování standardních funkcí i18n, jako jsou množná čísla, kontext, interpolace či formát. Poskytuje také kompletní řešení pro lokalizaci webých, mobilní či desktopových aplikací.

4 Pracovní náplň

Ve firmě ATLAS Consulting s.r.o. [1] jsem měl možnost podílet se na vývoji webového frontendu, spolupracovat v menším týmu. Mou hlavní náplní práce bylo, analyzovat problémy a navrhovat jejich řešení, využívat nejmodernější technologie, které se využívají ve webových aplikacích, podávání zpětné vazby ke zdrojovým kódům svých nynějších kolegů a testování jednotlivých výstupů. Znatelná část odborné praxe byla úzce spojena s vývojem webové prezentační vrstvy pro software Codexis [2]. Hlavní náplň práce byla rozšířit tento projekt o novou funkcionalitu či zobecnit a zjednodušit zdrojový kód se stávající funkcionalitou. Převážně se jednalo o vytvoření nové, moderní prezentační vrstvy a udržitelného, znovupoužitelného zdrojového kódu. Díky knihovny React bylo možné rozdělit jednotlivé části webové prezentační vrstvy do ucelených, znovupoužitelných modelů, které jsou nazývané jako komponenty. Tyto komponenty je možno využívat napříč celou škálou projektů, které využívají technologii React, tyto projekty se úzce svážou s balíkem komponent a vzniká na modulu závislost, díky tomuto přístupu je možno zapouzdřit a sjednotit logiku webové prezentační vrstvy.

4.1 Průběh a organizace vývoje

V průběhu odborné praxe jsem byl součástí týmu, který se zabýval vývojem softwaru Codexis, je rozdělen na dva menší celky. Jeden celek se zabývá vývojem backendové části aplikace v Javě a druhý, jehož jsem byl součástí, řešil problematiku webového frontendu pro uvedený software. Komunikace a spolupráce mezi těmito dvěma celky byla opravdu klíčová. Backendový tým tvořila menší skupina lidí, která řešila problematiku s daty dokumentů, judikátů a legislativy. Frontendový tým se skládal ze čtyř lidí. Grafik, který vytvářel grafické návrhy a také se věnoval problematice kaskádových stylů, ostatní členové týmu, včetně mě, byli zaměřeni na programování v jazyce TypeScript s využitím technologie React. Vývoj v tomto týmu fungoval agilně, po jednotlivých iteracích probíhaly konzultace, revize zdrojového kódu a samotné implementace jednotlivých úkolů. Úkoly, na kterých jsem měl pracovat, zadával vedoucí týmu. Tyto úkoly byly předem domluveny s vedením společnosti, podle priority a požadavků, které byly důležité pro neustálý pokrok daného projektu. Na základě specifikací a předem domluvených kritérií daných úkolů byl vytvořen plně responzivní grafický návrh, který využíval standardní vzhled komponent, které byly naimplementované v knihovně komponent. Následovala analýza a návrh funkcionality ze strany webového frontendu, problémy jsme mezi sebou konzultovali, výstupem pak byl návrh řešení daného problému. Samotná implementace byla vždy v souladu s konzultacemi s kolegy, jestliže nastal nějaký problém, který se nepodařilo předem správně zanalyzovat, jako například chyba v použité knihovně třetí strany, pak proběhla další iterace a upravila se specifikace úkolu. Před tím, než se nová funkcionalita vypustila do produkční verze, bylo třeba tuto funkcionalitu otestovat pomocí testování založeném na principu konec-konec, kde se testovala funkčnost z pohledu běžného uživatele aplikace Codexis.

5 Řešení zadaných úkolů

Úkoly byly cíleny převážně na aplikaci Codexis [2], kde bylo aktuální přepsání starého zdrojového kódu webové prezentační vrstvy do nové, moderní verze, která bude plně responzivní, otestovaná a lehce udržovatelná. Bylo nutné řízení původních statický CSS souborů, postupného zapouzdřování zdrojového kódu, jenž obsahuje implementace a funkcionalitu knihovny Redux a následné nahrazení této funkcionality za modernější a jednodušší způsob s využitím React Context [8] a platformy Apollo [10]. V následující kapitole přiblížím specifikaci, analýzu a řešení jednoduchých dílčích úkolů, pro komplexní a více časově náročné úkoly budou vyhrazeny samostatné kapitoly.

5.1 Časová náročnost jednolivých úkolů

- Responzivita [2 dny]
- Modální a vyskakovací okna [2 dny]
- Animace [2 dny]
- Vícejazyčnost [1 den]
- Řízení původních CSS [1 den]
- Widget [2 dny]
- Knihovna komponent [7 dní+]
- Nová implementace původních záložek aplikace Codexis [30 dní]
- Projekt X [14 dní]

5.2 Responzivita

Specifikace a požadavky

Požadavky na tento úkol byly zřejmé, vytvořit plně funkční, responzivní chování pro aplikaci Codexis. Z toho důvodu, aby byla aplikace jednoduše použitelná na mobilních zařízeních a v různých rozlišeních.

Analýza a návrh

Velkým problémem responzivního chování v aplikacích, jenž používají knihovnu React.js může být vysoký nárok na výpočetní výkon, kvůli vykreslování komponent se všemi vlastnostmi pro všechna zařízení a následného řešení přes CSS. Proto byl zvolen jiný způsob, díky kterému bylo možné řešit responzivní chování na úrovni programovacího jazyka JavaScript. Obrovskou

výhodou v tomto principu byla možnost volání funkcí a vykreslování jednotlivých komponent do DOMu prohlížeče právě pro zvolená rozlišení a nebylo třeba spouštět nadbytečný kód a vykreslovat komponenty, které by pro uživatele nebyly viditelné.

Implementace

Na základě požadavků a analýzy tohoto problému bylo zvoleno řešení pomocí detekce vlastností prohlížeče, z objektu window, konkrétně rozlišení, tyto výsledné hodnoty byly použity v komponentě Device, jež poskytovala React Context a informovala posluchače o tom, které zařízení právě využívá aplikaci. S toutou logikou následně pracovala komponenta s názvem Device, která na základě specifikovaného zařízení vrátila její obsah.

Časová náročnost

Vyřešení tohoto problému mi zabralo jeden pracovní den.

5.3 Modální a vyskakovací okna

Specifikace a požadavky

Úkolem bylo naimplementovat komponentu, která dokáže vykreslit modální či vyskakovácí okno s libovolnými parametry a obsahem na jakémkoli místě.

Analýza a návrh

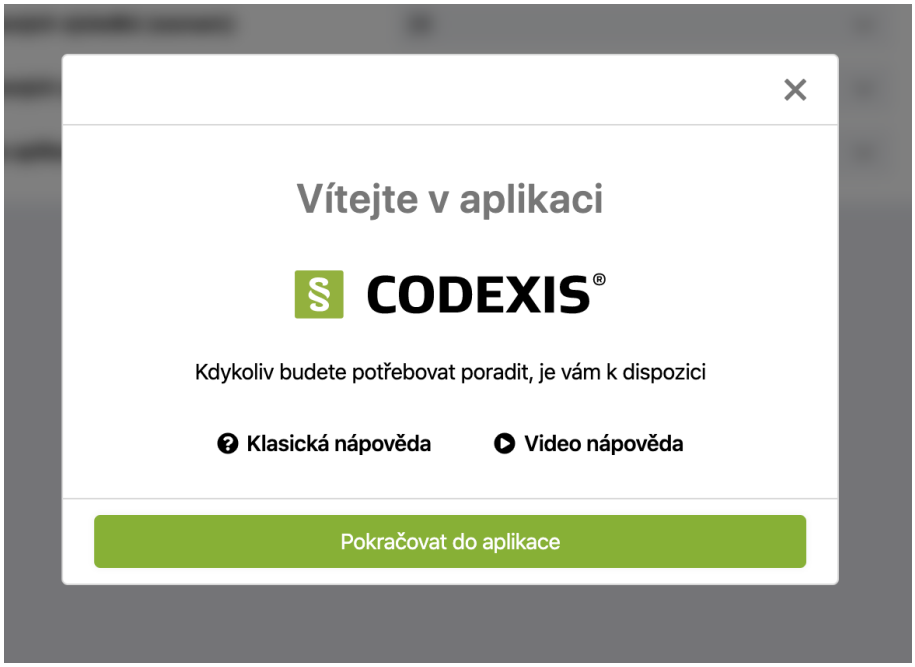
Na základě těchto abstraktních požadavků bylo možno zvolit více řešení tohoto problému. Bylo nutno rozlišit modální okna, která za sebou měla vykreslený overlay a vyskakovací okna (takzvaný Popup), který se měl dokázat vykreslit v jakémkoli HTML elementu.

Implementace

Na vyřešení tohoto úkolu byla použita technologie React Context, díky které bylo možno jednoduše informovat konkrétní komponenty o tom, zda se má při nějaké určité akci vyvolat otevření či zavření modálního okna. Tento ModalContext poskytuje metody showModal pro zobrazní modálu a hideModal pro jeho skrytí. Metoda showModal, která přijímá dva parametry, prvním je komponenta, kterou modal má vykreslit, druhý parametr je volitelný, kde je možné specifikovat rodiče, element, do kterého se má první parametr vykreslit, díky čemuž jsme schopní vytvořit vyskakovací okno, které není modální, výchozí hodnota je element HTML Body. 1

Shnutí

První den probíhala příprava jednotlivých komponent pro funkcionalitu vyskakovacích oken tak, aby se komponenty dokázaly vykreslit kdekoli v aplikaci. Druhý den probíhalo testování a optimalizace.



Obrázek 1: Modální okno

5.4 Animace

Specifikace a požadavky

Zadáním bylo vytvořit libovolné, svižné a moderní animace v aplikaci Codexis, funkčním požadavkem bylo využít technologii React Spring, jež řeší tuto problematiku.

Analýza a návrh

S touto knihovnou bylo možno podchytit většinu běžných animací, které byly využity v modálních oknách a jiných komponentách jako Slider či Carousel.

Implementace

Implementace byla různá, záleželo na konkrétním problému, bylo možno využívat předdefinované vlasnosti uvedené knihovny, nebo tvořit animace vlastní.

Časová náročnost

Jelikož se implementace lišila podle potřeby specifických animací na konkrétním místě, nedá se přesně časová náročnost určit. Příprava pro fungování animací v projektu zabrala jeden den, jednotlivá využití těchto animací byly vyřešeny v rámci několika málo hodin.

5.5 Vícejazyčnost

Specifikace a požadavky

Za úkol bylo v aplikaci Codexis naimplementovat funkčnost pro více jazyků, tak aby statické konstanty, které se vykreslují na webovém frontendu aplikace bylo možno překládat.

Analýza a návrh

Na řešení tohoto úkolu byla zvolena technologie i18next, knihovna, která se touto problematikou zabývá v celém světě JavaScriptu.

Implementace

Nástroj i18next je naimplementován taky pro technologii React, služba poskytuje komponentu, která pomocí React Context řídí a naslouchá na použitých vlastnostech a dokáže reagovat na změnu jazyka, poskytuje také metodu translate, která přijímá jako vstupní parametr překladovou konstantu. 1

```
const TranslationProvider: FC<PropsWithChildren<ContextProps>> = (props):
    ReactElement => {
  const { children } = props;
  const { i18n, t } = usei18nextTranslation();
  const setLanguage = async (language: Language) => await i18n.changeLanguage(
      language);

  return (
    <I18nextProvider i18n={i18n}>
      <TranslationContext.Provider
        value={{
          language: i18n.language as Language,
          t,
          translate: t,
          setLanguage,
        }}
      >
        {children}
      </TranslationContext.Provider>
    </I18nextProvider>
  );
```

Výpis 1: Komponenta pro řešení překladů

Shrnutí

Překlady samotné existují ve statických souborech, které se doplňují na základě specifikace a potřeby. Časová náročnost návrhu a implementace byla jeden pracovní den.

5.6 Řízení původních CSS

Specifikace a požadavky

Nutným požadavkem pro jednoduchou práci bez žádných vedlejších efektů bylo vypnutí původní implementace kaskádových stylů, v nově naimplementované záložce aplikace, který by mohli potencionálně rozhodit rozložení stránky či změnit design našich zapouzdřených předvytvořených komponent z knihovny React Atlantic.

Analýza a návrh

Nutnou úpravou bylo odstranit vazbu na statický soubor, jenž tyto styly obsahoval, v konkrétní záložce aplikace,tak aby po přepnutí na jinou záložku, byl tento styl stále zachován.

Implementace

Problém se statickým souborem byl vyřešen pomocí komponenty, která uvnitř využívá funkcionalitu React Context, kterou poskytuje knihovna React. Na základě informace o aktivní záložce aplikace bylo možné rozhodnout, kdy tento kaskádový styl odebrat či přidat zpět do HTML dokumentu. S využitím React hooků bylo možné při načtení komponenty, která reprezentuje konkrétní záložku aplikace předat boolean informaci konkrétnímu Contextu o tom, zda má být styl zobrazen či nikoli. Na tomto Observeru naslouchá komponenta, která nedělá nic jiného než odebírá či přidává do HTML hlavičky element link s odkazem na původní kaskádové styly.2 Časová náročnost implementace tohoto úkolu byla jeden pracovní den.

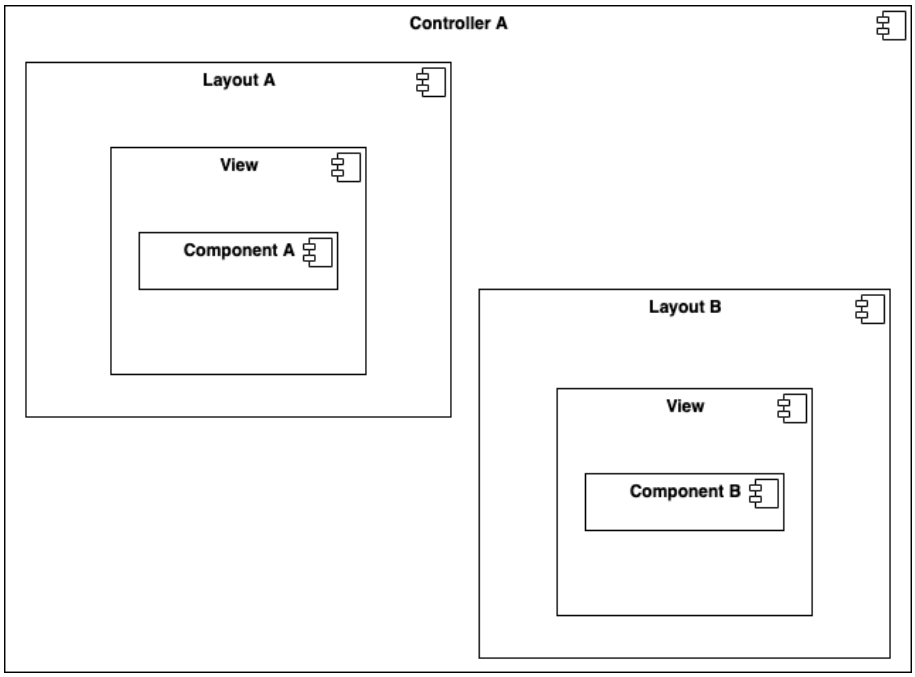
```
const { setOldStyleVisibility } = useStyleRefactor();
   useEffect(() => {
      setOldStyleVisibility(false);
      return () => setOldStyleVisibility(true);
   });
const OldStyle: FC = () => {
   const { isOldStyleVisible } = useStyleRefactor();

   if (isOldStyleVisible) {
      return (
         <Helmet>
            <link rel="stylesheet" type="text/css" href="dist/style.css" />
         </Helmet>
```

```
      );
    }
    return <GlobalStyle />;
};
```

Výpis 2: Příklad odebírání a přidávání původních CSS

5.7 Widget

8



Obrázek 2: Architektura Widget komponent

Specifikace a požadavky

Na základě znovupoužitelnosti složitějších komponent, které byly v aplikaci Codexis označovány jako Widget a byly složeny právě z komponent z knihovny React Atlantic, bylo nutné vymyslet strukturu zdrojového kódu tak, aby nevznikly přebytečné duplicity v byznys logice aplikace.

Analýza a návrh

Jednotlivé Widget komponenty byly rozděleny na tři vrstvy, Controller, Layout a View, kde Controller řešil problematiku se samotnými daty, layout rozložení Widgetu v konkrétní implementaci a View poskytoval komponenty pro daný Widget. Analýza tohoto problému a návrh trvala více než jeden pracovní den.

Implementace

Controller byl napojený na React Context komponenty a získával z nich data, které byly zapotřebí vykreslit uživateli v dalších vrstvách Widget komponent. Layout komponenty využívaly Device Context a řešilo se v nich rozložení na specifickém místě v aplikaci, vykreslovaly komponenty, které poskytuje View. View vrstva vracela objekt, ve kterém byly uloženy jednotlivé komponenty, které tento Widget poskytuje. Implementace samotná oproti návrhu nebyla vůbec složitá, její časová náročnost byla několik jednotek hodin.

6 Knihovna komponent (React Atlantic)

Specifikace a požadavky

Hlavním úkolem bylo vymyslet řešení pro stabilní vývoj produktu Codexis a dalších projektů , které fungují na webové prezentační vrstvě. Na základě předchozích zkušeností s tímto projektem a zkušenostmi se zvolenými technologiemi bylo potřeba rozdělit tento software do jednotlivých modulů a jedním z nich byla právě knihovna komponent, React Atlantic. Tyto komponenty měli být možné využívat v jakékoli aplikaci, která využívá technologii React.
Funkční požadavky: znovupoužitelnost, atomičnost, izolovanost, modularita a dokumentace.

Analýza a návrh

Vyřešení tohoto úkolu bylo jedno z nejpodstatnějších pro další vývoj projektu Codexis. Na základě specifikací a požadavků bylo možné zdrojový kód knihovny psát plně otevřený pro veřejnost. Knihovna měla poskytovat balík znovupoužitelných, atomických komponent, které jsou plně otestované, parametrizovatelné a jejich stav je zapouzdřen do jednotlivých funkcí. Velkou insiprací pro tento balík, byly knihovny jako AntDesign či MaterialUI. Tyto komponenty mezi sebou měli sdílet informace v reálném čase, proto bylo vhodné zvolit návrhový vzor Observer, díky kterému dokázaly reagovat jedna na druhou, podle potřeby.

Implementace

Knihovna komponent byla hostována v repozitáři ve službě GitHub, napsána v jazyku TypeScript a využívala React pro vykreslování jednotlivých komponent. Pro izolování kaskádových stylů byla zvolena technologie CSS-in-JS, konkrétně knihovna Styled Components, která umožnila jednoduchý a bezpečný přístup zapouzdření stylů do komponenty. Komponenty jsou plně otestovány pomocí jednotkových testů, na které byla zvolena technologie Jest a Enzyme. Pro statickou dokumentaci těchto komponent byl zvolen nástroj jménem Storybook, který umožnil vytvořit jednoduché a přehledné prostředí pro komponenty, ve kterém má uživatel k dispozici výčet všech komponent, které tato knihovna poskytuje, taktéž jejich statické typy, vlastnosti a stav, které komponenty poskytují.

Využití v aplikacích

Primární byly tyto komponenty tvořeny na základě grafických podkladů a specifikací pro aplikaci Codexis[2]. Grafik, který tyto komponenty kreslil, dodržoval určité interní standardy, které se ve společnosti nastavily, což umožnilo velkou znovupoužitelnost těchto grafických komponent. Ve většině záložek aplikace Codexis již byla naimplementována podpora pro tyto komponenty, dále komponenty také byly využity v aplikaci Project X nebo například na webovém portálu Právní Prostor[13]. Tyto komponenty se v budoucnu budou rozšiřovat, zlepšovat a budou se obohacovat

o vlastnosti, podle potřeby daného softwaru. Dalším projektem, který by knihovnu komponent měl využívat, je projekt Hlídač ochraných známek [14] a všechen ostatní nově vytvářený software v této společnosti, který bude využívat technologii React. V následujících bodech budou více přiblíženy mnou vytvořené zajímavé komponenty:

6.1 Tabs

Specifikace a požadavky

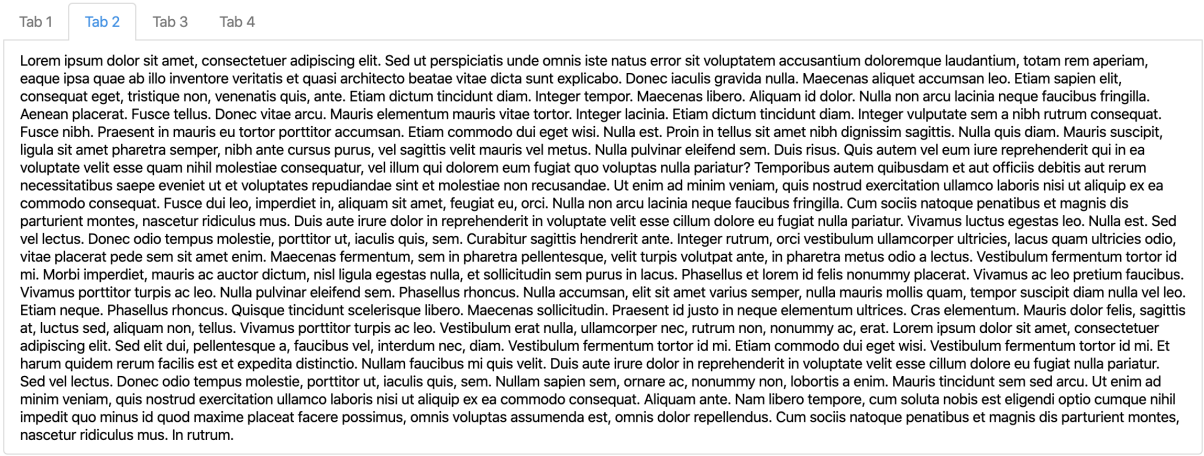
Komponenta Tabs měla umět vykreslit libovolný počet záložek a k nim specifický obsah, měla být také plně přístupná přes klávesnici, pomocí šipek vlevo, vpravo.

Analýza a návrh

Na základě jednoduché specifikace bylo zřejmé, že uvedená komponenta bude mít jednu z vlastností množinu prvků, které reprezentovaly jednotlivé záložky a obsah. Nutná byla také informace o aktivní záložce. Jednotlivé prvky této množiny byly reprezentovány jako React komponenty, byla využita kompozice mezi komponentou Tabs a množinou komponent Tab.

Implementace

Jednotlivé záložky jsou implementovány jako input typu radio, který je schovaný pomocí kaskádových stylů a kontrolován pomocí stavu komponenty. Pro komunikaci mezi jednotlivými Tab komponentami jsem zvolil funkcionalitu React Context [8], díky které lze jednoduše předávat informace o aktivní záložce do komponenty Tab a manipulovat s funkcemi na úrovni Tabs komponenty. Komponenta přijímá vlastnosti, které implementují následující rozhraní: 33



Obrázek 3: Tabs

```
interface TabsProps<T> {
  onChange: (activeTab: Readonly<T>) => void;
  animationConfig?: Readonly<SpringConfig>;
  activeTab?: Readonly<T>;
  tabs: Readonly<TabProps> | Readonly<TabProps>;
  size?: Readonly<Size>;
  className?: Readonly<string>;
  isBordered?: Readonly<boolean>;
  isAlternative?: Readonly<boolean>;
}
```

Výpis 3: Vlastnosti komponenty Tabs

6.2 Table

Specifikace a požadavky

Úkolem bylo vytvořit komponentu, jež dokáže vykreslit obecnou tabulku ve specifickém vzhledu, komponenta také musela umět přijmout pouze komponenty, ze kterých se skládala, všechny ostatní měla ignorovat. Tyto komponenty měly být hlava, tělo, řádek a sloupec.

Analýza a návrh

Tabulka byla rozdělena na jednotlivé komponenty, ze kterých se podle specifikace měla skládat, proto bylo zapotřebí vytvořit komponentu Table, Table.Head, Table.Body, Table.Row a Table.Column, tyto komponenty mezi sebou měly umět komunikovat podle návrhového vzoru Observer.

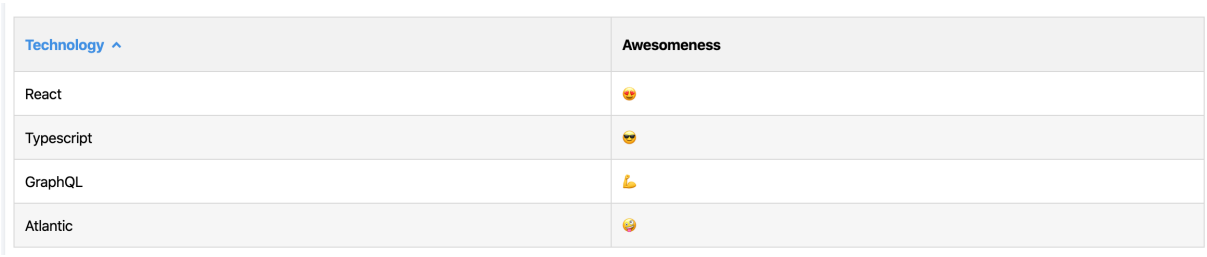
Implementace

Pro implementaci byla zvolena kompozice Tabulky z předem navržených komponent. Komponenta Table se skládala z komponent Table.Head a Table.Body, do těchto komponent bylo možné vkládat pouze komponenty řádku, Table.Row. Komponenty řádku se skládaly pouze z komponent sloupců, Table.Column. Jednotlivé sloupce dokázaly vykreslit libovolný obsah. Díky této striktní kompozici bylo jednoduché dodržet konkrétní vzhled tabulky.
Vlastnosti tabulky implementují tento interface: 4 4

```
interface TableProps {
  onSort?: (sortBy: Readonly<string>, order: Readonly<SortOrder>) => void;
  sortBy?: Readonly<string>;
  order?: Readonly<SortOrder>;
```

```
}
```

Výpis 4: Vlastnosti komponenty Table



Obrázek 4: Table

6.3 Timeline

Specifikace a požadavky

Komponenta časové osy, která měla umět vykreslit seznam komponent s nadpisem, popisem a libovolným počtem ikon.

Analýza a návrh

Pro vykreslení seznamu komponent bylo nutné vytvořit jednotlivé položky pro tento seznam, Timeline.Item, které komponenta Timeline bude umět vykreslovat. Bylo nutné ukládat informaci o aktivní položce a také detekovat změnu aktivní položky.

Implementace

Implementace spočívala v tom vytvořit komponentu Timeline.Item, která měla naslouchat na React Context, který poskytovala komponenta Timeline, díky tomu byla možná detekce aktivní položky a využívání vlasností, které komponenta uměla přijímat. Rozhraní této komponenty vypadá následovně: **??** 5

```
interface TimelineProps {
  onChange?: (index: Readonly<number>) => void;
  activeIndex?: Readonly<number>;
  defaultActiveIndex?: Readonly<number>;
}
```

Výpis 5: Vlastnosti komponenty Timeline

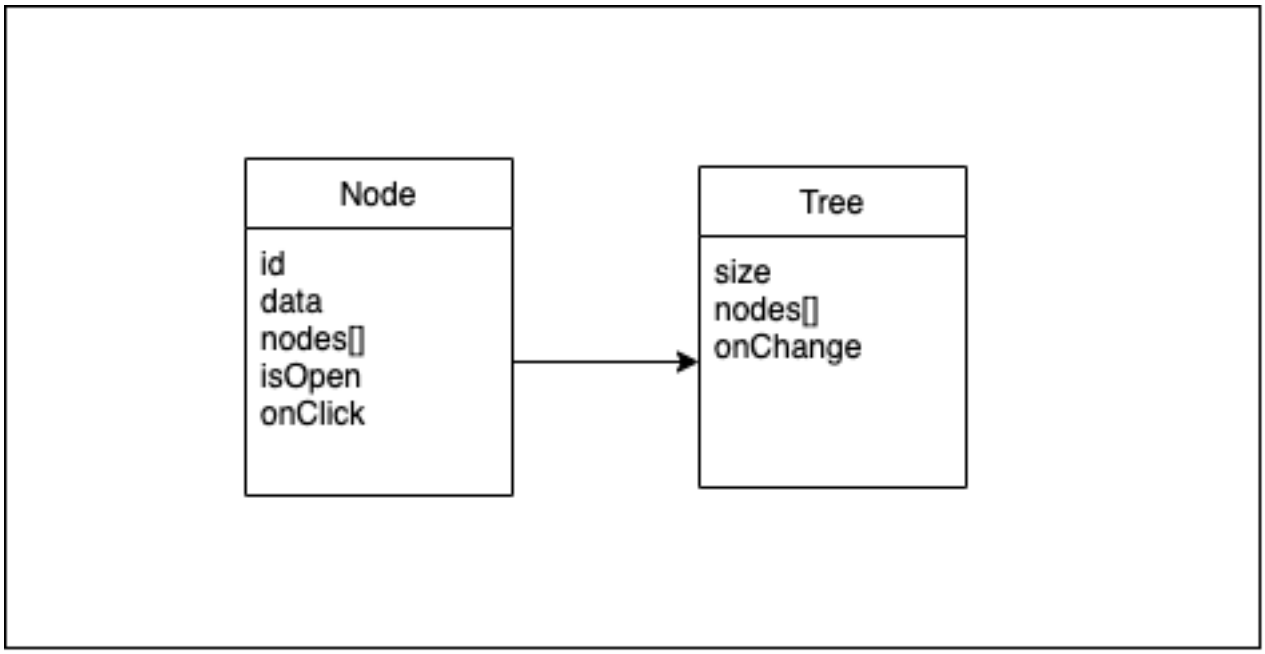
6.4 Tree

Specifikace a požadavky

Zadáním bylo vytvořit komponentu abstraktního stromu či hierarchického seznamu. Tato komponenta měla umět pro jednotlivý uzel vykreslit libovolný obsah. Jedná se o logickou i grafickou komponentu zároveň. Aplikaci Codexis tvoří z velké části grafické zobrazení jednoduchých hierarchických listů, proto bylo nutné mít plnou kontrolu nad těmito komponentami, další podmínkou byla práce pomocí JSX zápisu.

Analýza a návrh

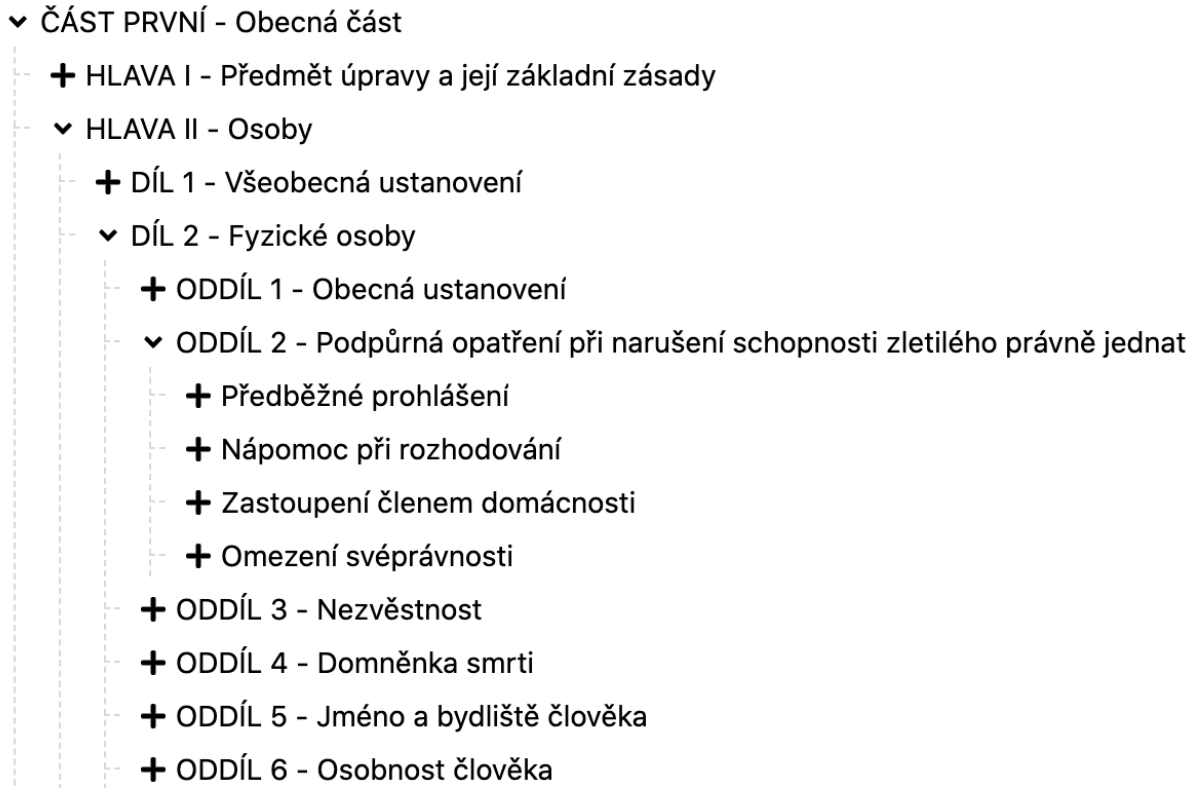
Komponenta byla navržena tak, aby vyhovovala specifikaci zadání a grafickým návrhům. Vlastnosti, které základní komponenta obsahuje byly pouze primitivní, jako vykreslení alternativního vzhledu a jeho grafické velikosti. Jednotlivé uzly musely obshovat metody na změnu, klikání a jiné posluchače událostí, ikonu uzlu a generický parametr data. Bylo zvoleno vlastní řešení této komponenty, z důvodu poměrně jednoduché specifikace a vlastních grafických návrhů. Důležitým faktorem pro tuto komponentu byla mimo jiné úplná kontrola nad zdrojovým kódem, díky které lze jednoduše rozšiřovat komponentu o další potřebnou funkcionalitu. Nebylo nutné využití komponenty třetí strany z důvodu jednoduchosti a konkrétnosti zádání.5



Obrázek 5: Tree

Implementace

Byly vytvořeny komponenty Tree a Tree.Node. Tree se skládala z listu Tree.Node a vykreslovala je. Tree.Node komponenta mohla přijímat libovolný objekt v argumentu children. Jestliže obsahem byla komponenta uzlu, pak se znovu rekurzivně vykreslila komponenta Tree s těmito uzly. 6



Obrázek 6: Příklad implementace komponenty Tree

6.5 Transfer

Specifikace a požadavky

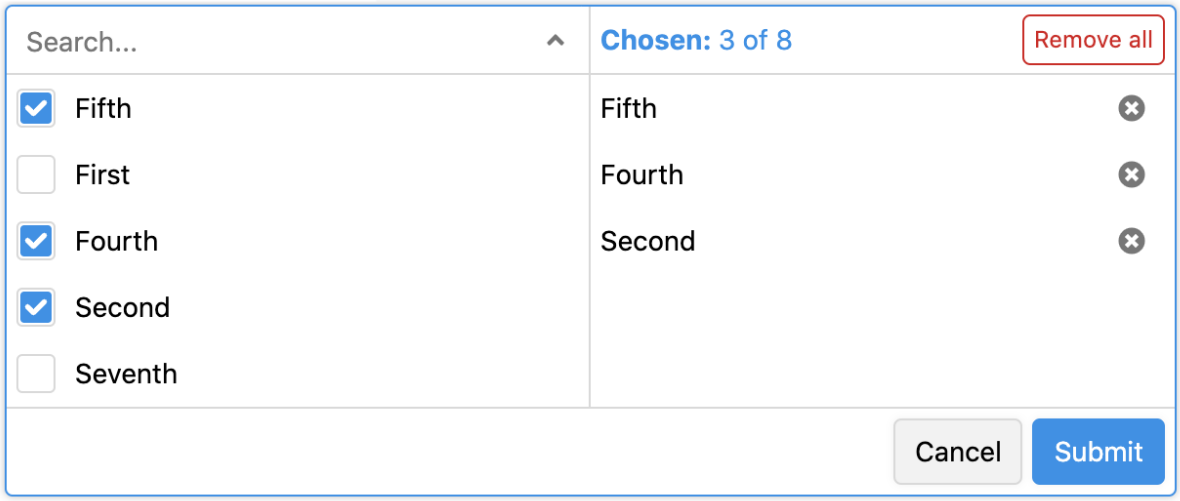
Inspirací pro tuto komponentu byl takzvaný MultiSelect, jedná se o obdobu této komponenty, uživatel je schopen zvolit jednu či více možností z množiny předem specifikovaných možností. Zadáním bylo tedy vytvořit tuto komponentu tak, aby uživatel mohl z jedné strany převést možnosti na druhou a uložit či stornovat vstup. V možnostech bylo možno také vyhledávat, pomocí podřetězce. Bylo možno také odebrat všechny zvolené možnosti.

Analýza a návrh

Tato komponenta byla navržena tak, aby se dala sestavit z již předem vytvořených tlačítek, zaškrtávacích políček a typografie. Bylo nutno navrhnou komponentu tak, aby bylo možno využívat syntax JSX. Jednotlivé možnosti obsahovaly text a hodnotu.

Implementace

Komponenta Transfer se skládala z listu komponent Transfer.Option, tato komponenta obshaovala dva parametry, label a value, kde label byl vykreslen do jednotlivé části komponenty a podle parametru value se rozlišovaly jednotlivé možnosti. Bylo nutné ukládat hodnoty jednotlivých vybraných možností ve stavu komponenty Transfer. Možnosti s odpovídajícími hodnotami se poté vykreslovaly do pravé části komponenty. 7



Obrázek 7: Grafická ukázka komponenty Transfer

Shrnutí

Přesná časová náročnost tohoto úkolu nelze přímo určit, jednotlivé komponenty byly vytvářeny na základě grafických návrhů, dle iniciativy grafika, průměrná doba strávená na vytvoření jedné komponenty byla dva dny. Analýza, návrh, příprava a samotná implementace projektu pro tvoření těchto komponent zabrala zhruba jeden pracovní týden, kde první dny probíhala analýza, výběr specifických technologí a návrh adresářové struktury projektu, příprava projektu pro jednotkové testování zabrala jeden den, časová složitost implementace generování statické dokumentace zabrala jeden den.

7 Nová implementace původních záložek aplikace Codexis

Specifikace a požadavky

Po vytvoření knihovny komponent a připavených řešení na úpravu aplikace, bylo za úkol, s využitím této knihovny, přepsání a zjednodušení zdrojového kódu pro jednotlivé záložky aplikace Codexis. Časová náročnost jednotlivých záložek se pohybovala okolo dvou až tří týdnů, kde v prvním týdnu probíhal přepis starého kódu za nový, v druhém týdnu týdnu byla řešena oprava chyb a v poslední fázi vývoje probíhalo testování těchto záložek. Funkční požadavky: Využití knihovny React Atlantic, dodržení responzivních grafických podkladů a zachování původní funkcionality aplikace.

Analýza a návrh

Nejjednodušší způsob pro dodržení všech uvedených aspektů, byl odstratit původní zdrojový kód, který byl nahrazen novým, jenž implementoval veškeré specifikace.

Implementace

Veškerá původní implementace záložek, byla tedy smazána a nahrazena novým zdrojovým kódem, ve kterém bylo za úkol využívat již zmíněnou funkcionalitu responsivního chování, vícejazyčnosti a knihovnu komponent.

Globální komponenty aplikace

Před tím, než začal probíhat přepis jednotlivých záložek aplikace Codexis, bylo nutné přepsat původní implementace globálních prvků a komponent aplikace, které se vyskytují v aplikaci pokaždé, mimo jednotlivé záložky. Hlavička aplikace obsahovala tlačítka pro řízení aplikace, jako je hlavní nabídka, datum aplikace či profil uživatele, také obsahuje list s otevřenými záložkami Patička, která se vykreslovala pouze na mobilních zařízeních, která v sobě obsahovala různé řídící prvky, které sloužily pro jednoduchou práci s aplikací na nižším rozlišení.

Titulní strana

Titulní strana aplikace se skládala z pěti níže uvedených Widget komponent, kde každá Widget komponenta řešila specifickou problematiku přehledného zobrazování informací uživateli na titulní straně.

1. Widget Search
2. Widget History
3. Widget Favourites

4. Widget Followed

5. Widget News

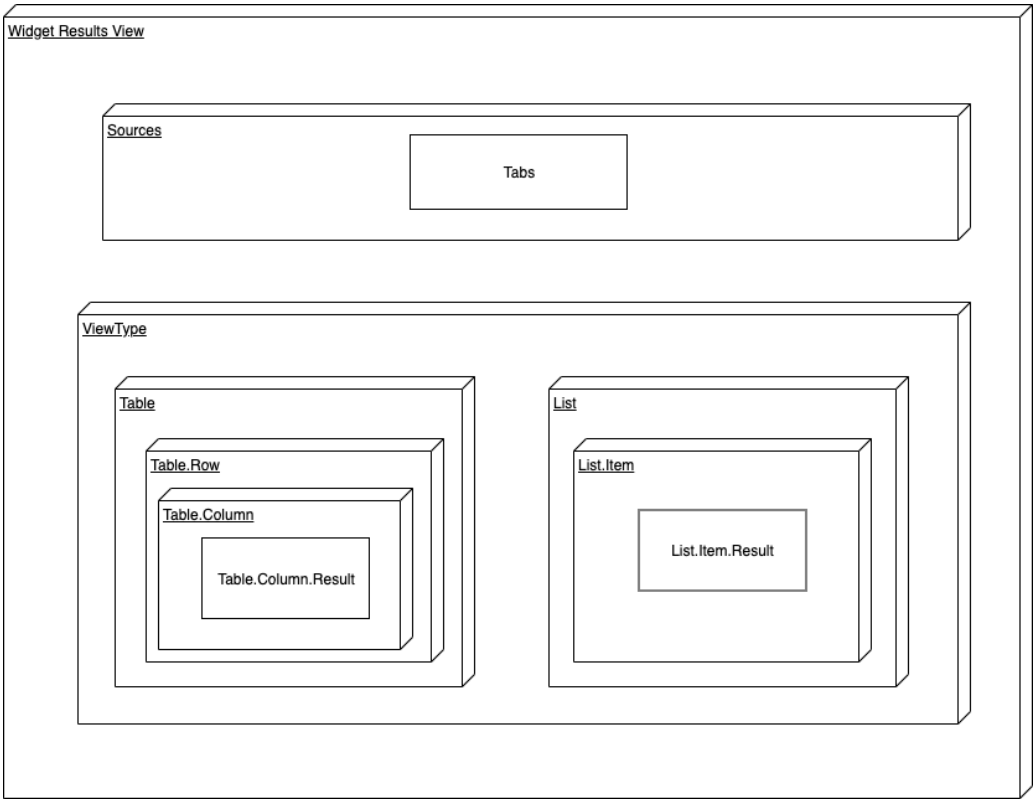
Komponenta Widget Search se využívala na více místech v aplikaci, proto byla naimplementovaná jako Widget, komponenta která má tři vrstvy, využívala se taktéž například v záložce vyhledávání nebo dokumentu. Widget ListDocument sloužil k vykreslení jakéhokoli seznamu s dokumenty, které se v aplikaci Codexis vyskytují, je například využití v komponentách jako jsou Widget History, Widget Favourites či Widget Followed. 13 Widget News se zabýval vykreslováním novinek a aktualit z webového portálu Právní Prostor[13], GraphQL server poskytoval ve svém schéma metodu, která vracela ve specifickém formátu data stažena z RSS kanálu a díky striktnímu typovému systému z GraphQL server tato data musí staticky otypovat, díky čemuž klientská aplikace dokáže jednoduše zjistit, v jakém přesném formátu tato data jsou. Pomocí nástroje Apollo Client pro React a využitím Apollo Hooks má tento Widget napojený Controller přímo na tento GraphQL dotaz.

Vyhledávání

Záložka se zabývala vyhledáváním dokumentů, byla složena ze třech základních Widget komponent, Widget Search pro vyhledávání a manipulaci s výsledky, Widget Results pro vykreslení samotných výsledků vyhledávání a Widget Filtration, který obsahoval parametrická pole, dle kterých lze filtrovat výsledky vyhledávání. Widget Filtration, jenž dával uživateli k dispozici parametrická pole, podle kterých bylo možno filtrovat výsledky vyhledávání.15

Widget Results

Komponenta Widget Results vykreslovala výsledky vyhledávání, které server poskytoval na základě GraphQL požadavku, který podle zadaných parametrů klientská aplikace dotazovala. View této Widget komponenty přijímá jako povinné parametery, volbu zobrazení, bylo možno volit mezi jednoduchým seznamem či tabulkou, tato možnost byla implementována jako enum. Dalším paramametrem je seznam datových zdrojů a aktivní datový zdroj. Komponenta také umí dynamicky vykreslovat specifické sloupce na základě datového zdroje nebo umístění. Nejdůležitějším parametrem jsou potom samotné výsledky vyhledávání, které se ve view vykreslují. View část Widget komponenty byla složena z komponent Sources, což byla implementace abstraktní komponenty Tabs, z tabulky a jednoduchého listu. V tabulkovém zobrazení bylo možno jednoduchého řazení výsledků například podle jména nebo data vydání.



Obrázek 8: Návrh komponenty Widget Results

Dokument

Nejvíce komplexní záložka, kterou aplikace Codexis obsahovala. Sloužila k manipulaci s dokumentem, jeho zněními a souvisejícími dokumenty. Záložka byla složena ze sedmi Widget komponent. Widget DocumentInfo vykrosloval základní informace a metadata konkrétního dokumentu. Widget Timeline sloužila k zobrazení jednotlivých znění dokumentů v časové ose. Widget TableOfContent, jenž vykresloval obsah dokumentu. Jednotlivé části obsahu byly klikatelné a na základě informace, jež obsahovala každá část, mohly reagovat ostatní Widget komponenty, díky čehož šlo například docílit změny pozice v dokumentu. Widget RelatedDocuments zabývající se souvisejícími dokumenty, které byly s konkrétním dokumentem v nějakém vztahu, uvnitř tohoto Widgetu byl využitý Widget Results, pro vykreslení výsledků, tyto výsledky šlo taktéž filtrovat. Widget RelatedComments dokázal vykreslovat hierarchickou strukturu komentářů, jenž se vázaly ke specifickým částem dokumentu. Tento Widget také dokázal vykreslit Widget Results pro jednoduché lineární zobrazení komentářů pro tento dokument. Widget DocumentControls byl jednoduchá komponenta, která vykreslovala vstupní uživatelské prvky a tlačítka, která dokázala ovládat chování dokumentu.

Widget Document

Widget Document, měl umět vykreslit samotný HTML obsah dokumentu, jenž poskytovalo GraphQL API ze serveru. Toto HTML bylo v několika případech opravdu obrovské, proto bylo nutno vymyslet řešení takové, aby nebyl kladen důraz na vysoký výkon zařízení. Když se jednoduše vykreslil celý HTML obsah velkých zákonů, prohlížeč přestal reagovat. Původní řešení tohoto problému bylo implementováno pomocí knihovny třetí strany React-Virtualized, které řešilo vykreslení konkrétních částí HTML dokumentu, která se předpřipravila do jednotlivých segmentů. Toto řešení bohužel obsahovalo velké množství vykreslovacích problémů a grafických chyb, také nebylo možno v dokumentu vyhledávat pomocí nativního prohlížečového vyhledávače, jelikož nebyl v DOM vykreslený celý. Udělal jsem tedy návrh jiného řešení pro tento problém, využil jsem teoretických znalosti asynchronního a více vláknového programování, které jsem získal během studia na univerzitě a zvolil jsem způsob rozdělení jednotlivých segmentů daného dokumentů do jednotlivých skupin, algoritmus, který vykresloval tyto skupiny segmentů byl spušten ve vedlejším vlákně prohlížeče a i během vykreslování prohlížeč reagoval na interakce od uživatele, jelikož hlavní prohlížečové jádro nebylo plně vytíženo. Toto řešení mi bylo schváleno jako alternativní a lepší volba oproti původní implementaci pomocí knihovny React-Virtualized a také byla nad těmito algoritmy úplná kontrola. Algoritmus, který řešil průchod segmentů, jako první prochází všechny indexy v poli, které jsou před počátečním indexem, toto řešení bylo zvoleno z toho důvodu, protože během vykreslování jednotlivých částí dokumentu do HTML, se začal DOM zvětšovat a měnila se rolovací pozice. Proto bylo nutno rolování během iterace uživateli zablokovat.

```
const asyncLoopEdges = useCallback(
    callback: (
        index: Readonly<number>,
        callback: AsyncLoopCallback,
        delay: Readonly<number> = 0,
        preferBackwards: Readonly<boolean> = false
    ): ReadonlyArray<Readonly<NodeJS.Timeout | undefined>> => {
        let forward = index;
        let backward = index;

        const timeout = setTimeout( callback: () => loop(index), delay);
        let timeoutBackward: NodeJS.Timeout | undefined = undefined;
        let timeoutForward: NodeJS.Timeout | undefined = undefined;

        const loop = (iterator: Readonly<number>, isBackwards: Readonly<boolean> = true) => {
            callback(readonlyArray[iterator], iterator);

            if ((isBackwards || preferBackwards) && backward > 0) {
                timeoutBackward = setTimeout( callback: () => loop(--backward,  isBackwards: false), delay);
            } else if (forward < readonlyArray.length - 1) {
                timeoutForward = setTimeout( callback: () => loop(++forward,  isBackwards: true), delay);
            }
        };

        return [timeout, timeoutBackward, timeoutForward];
    },
    deps: [readonlyArray]
);
```

Obrázek 9: Algoritmus pro asynchronní průchod segmentů

8 Project X

Project X neboli právní poradna online je projekt, který byl vyvíjen během velmi krátkého časového intervalu, plně distančně. Na tomto projektu pracovalo zhruba 7 lidí. Tento produkt se začal prodávat za pouhé dva týdny od prvního napsáného řádku ve zdrojovém kódu.

Motiv

Hlavním motivem, a myšlenkou, proč tento projekt vytvořit, bylo spojení advokátních kanceláří s běžnými občany, kteří potřebují právní pomoc, nejjednodušším způsobem, jakým to bylo možné. Jedná se o platformu, která je poskytována právním kancelářím, které zde mohou komunikovat se svými klienty, následně bude tato platforma dostupná pro širokou veřejnost, kde bude kdokoli moci popsat svůj problém v jednoduchém uživatelském rozhraní, nahrát potřebné dokumenty či fotografie, komunikovat s vícero právníky pomocí běžného psaní nebo video hovoru, přijímat či odmítat nabídky a vzápětí je v této platformě přímo platit.

Specifikace požadavků

Jelikož projekt rostl opravdu rychle, vše bylo řešeno během několika dnů příprav a samotného vývoje, proto byla důležitá jednoduchá a přesná specifikace požadavků. V aplikaci mělo být umožněno přihlášení, pomocí emailové adresy či sociálních sítí. Rozdělení uživatelských rolí, na základě kterých je možno rozlišovat uživatelská oprávnění v aplikaci. Platforma byla založena na principu chatovacích místností, kde jedna místonost reprezentuje daný případ, který spolu uživatelé aplikace řeší. Zásadním požadavkem byla také komunikace přes hlasový hovor a video hovor. Aplikace měla poskytovat možnosti interaktivních prvků v chatu, jako například online platba kartou nebo různá rozhodovací tlačítka či nahrávání souborů, dokumentů a obrázků. Jestliže byl uživatel přihlášen v roli organizace, pak měl přístup do administrační části aplikace, kde mohl editovat informace o advokátní kanceláři či spravovat uživatele v této organizaci. Důležitým požadavkem bylo také jednoduché a intuitivní uživatelské rozhraní, jelikož se tato platforma měla dostat mezi širokou veřejnost lidí. Jeden z požadavků byl také na takzvaný branding, jelikož se jedná o platformu, mělo být umožněno plně customizovat loga, obrázky, texty a názvy v aplikaci pro jednotlivé společnosti, které tuto platformu využívají, v budoucnu případná integrace individuálních požadavků.

Technická specifikace

Jednalo se o webovou, cloudovou aplikaci, jež byla rozdeléna na klientskou a serverovou stranu. Tyto části spolu komunikovaly přes GraphQL API, které server poskytoval. Server byl postavený na technologii Java. Klientská aplikace byla napsána v jazyce TypeScript, vykreslování zajišťovala knihovna React. Velká část komunikace probíhala přes WebSocket protokol, konkrétně v implementaci GraphQL Subscriptions, díky kterému bylo možno jednoduše zasílat zprávy

mezi klientskou a serverovou stranou aplikace a využívat vlastnosti GraphQL, jako například statická datová typovost. Tento přístup dával možnost interaktivních reakcí, například jednoduchou obnovu zprávy či detekci online uživatelů. Hovory a video hovory byly řešeny pomocí technologie WebRTC a celý přenos datového toku, pro audio a video, probíhal přes technologii RTC. Byla zde také využita knihovna komponent React Atlantic, která ulehčila práci s jednoduchými komponentami v aplikaci, ať už se jednalo o vstupní prvky, tlačítka, ikony či typografii. Bylo nutné také upravovat zdrojový kód tohoto balíku, například z důvodu přidávání nových ikon, které v knihovně chyběly. Pro zjednodušení komunikace ze strany webové aplikace, byla zvolena platforma Apollo, v implementaci pro React.

9 Závěr

V bakalářské práci byla detailně popsána odborná praxe ve společnosti ATLAS consulting spol. s r.o. [1], kterou hodnotím velmi kladně. Byly představeny použité technologie, mezi nejdůležitější patří především JavaScript, TypeScript a React, které slouží primárně pro tvorbu webové prezentační vrstvy. Na základě přesných požadavků, specifikací a grafických návrhů byly řešeny jednotlivé problémy v aplikaci Codexis [2] a Project X. Komponenty, které byly vytvářeny během odborné praxe, byly nadále využívány v další projektech, například na webovém portálu Právní Prostor[13], kde tyto komponenty využívala externí softwarová společnost, která tento webový portál vytvářela.

9.1 Uplatněné znalosti

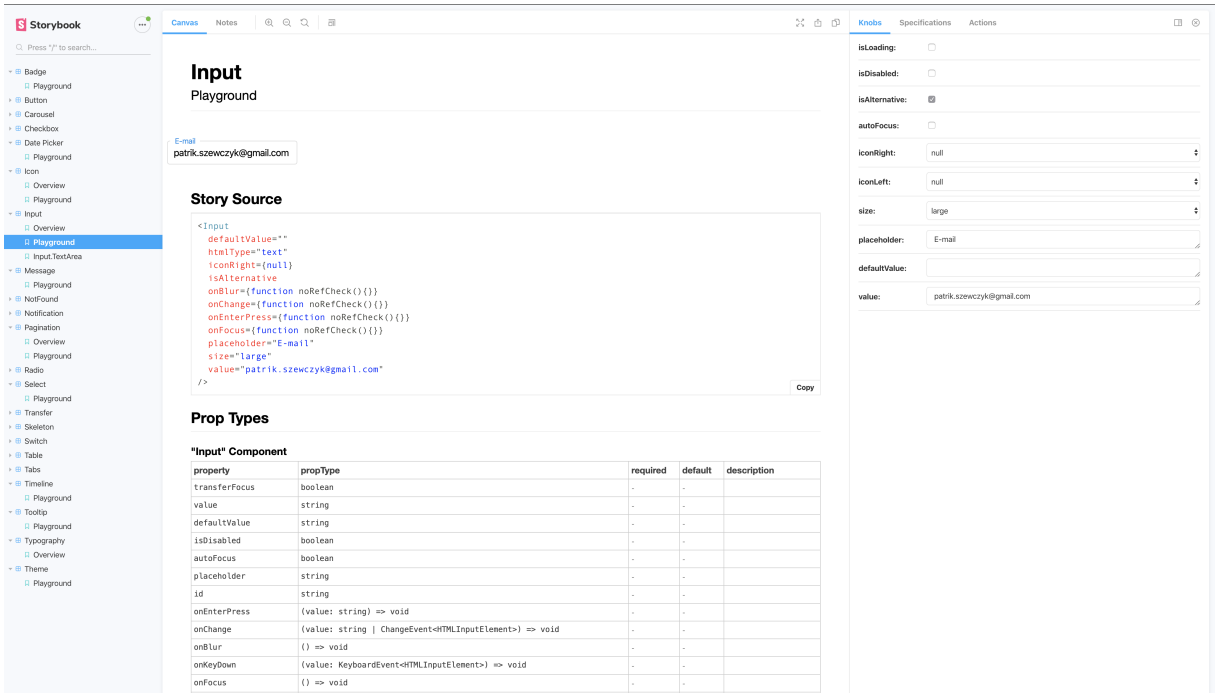
Mezi opravdu nejdůležitější uplatněné znalosti považuji ty teoretické, které jsem měl možnost získat během studia, jako jsou například znalosti algoritmizace, datových struktur a návrhových vzorů, jež jsem mohl během odborné praxe využít. Co se týče praktických znalostí, byly důležité znalosti jazyka JavaScript [3] a TypeScript [4], největším přínosem byl pro mě tedy předmět vývoj internetových aplikací, kde jsem tyto znalosti mohl získat. Důležitá byla také práce s verzovacími nástroji jako je Git, kterých jsem taktéž během studia dosáhnul, například v předmětu Programovací jazyky I a II.

9.2 Scházející znalosti

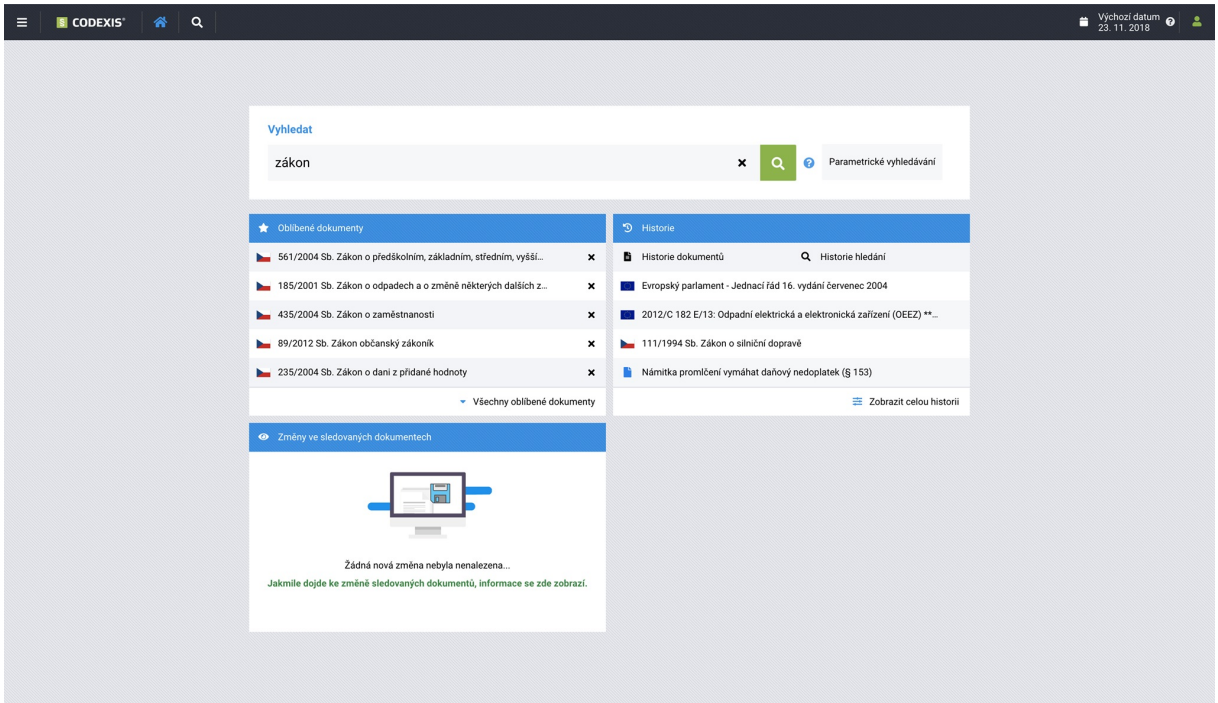
Znalosti získáné během absolbování odborné praxe beru jako velmi užitečné pro uplatnění na trhu práce v odvětví webového vývoje i mimo něj. Největší přínos pro mě byla technologie React a jeho moduly, se kterými jsem musel neustále pracovat. Jako další praktický přínos považuji práci s více lidmi najednou na jednom projektu, jejichž práce byla úzce spojena. Znalosti, které scházely nejvíce byly převážně spojeny s pokročilými, moderními, webovými technologiemi. Během studia jsem nezískal žádnou zkušenost s testováním zdrojového kódu, ať už se jedná o jednotkové či integrační testování. Také scházela znalost nástrojů pro průběžnou integraci, jako je například Jenkins.

Literatura

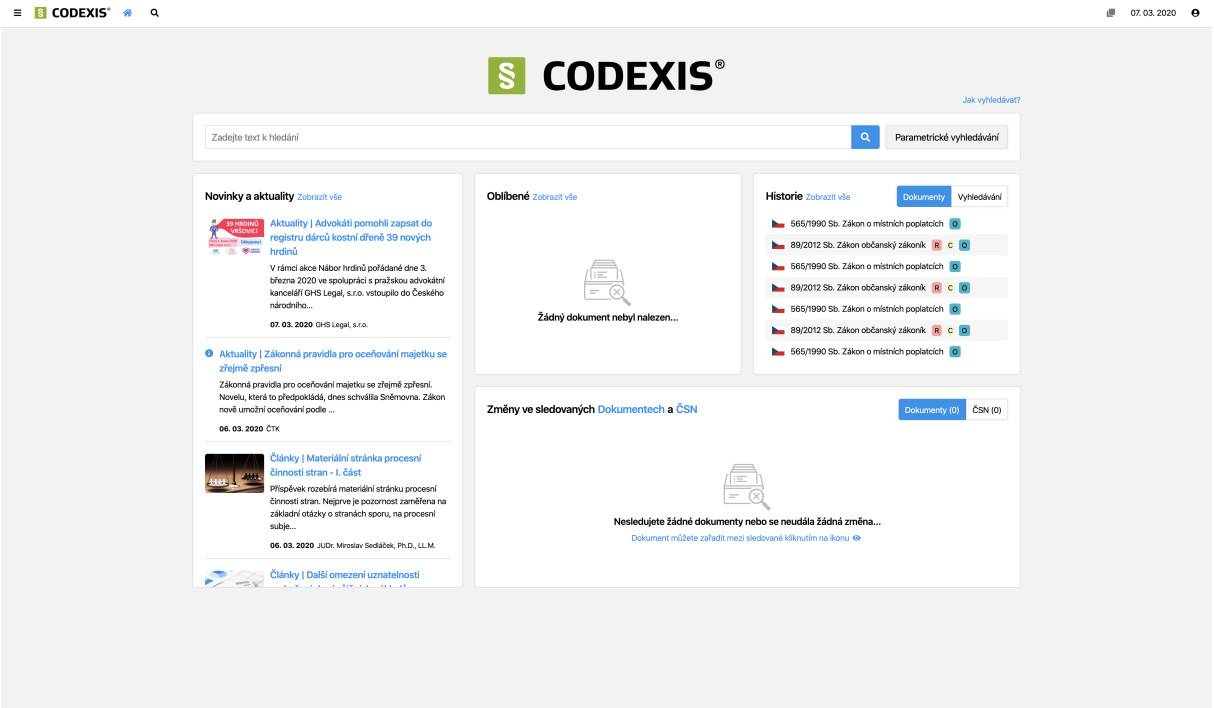
1. *Atlas Consuling s r.o* [online] [cit. 2020-03-07]. Dostupné z: https://atlasconsulting.cz/.
2. *Codexis* [online] [cit. 2020-03-07]. Dostupné z: https://atlasconsulting.cz/software/codexis/.
3. *JavaScript* [online] [cit. 2020-03-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.
4. *TypeScript* [online] [cit. 2020-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/TypeScript.
5. *React komponenty* [online] [cit. 2020-03-09]. Dostupné z: https://reactjs.org/docs/components-and-props.html.
6. *React stav a životní cyklus* [online] [cit. 2020-03-09]. Dostupné z: https://reactjs.org/docs/state-and-lifecycle.html.
7. *React hooks* [online] [cit. 2020-03-09]. Dostupné z: https://reactjs.org/docs/hooks-intro.html.
8. *React Kontext* [online] [cit. 2020-03-09]. Dostupné z: https://reactjs.org/docs/context.html.
9. *GraphQL* [online] [cit. 2020-03-10]. Dostupné z: https://graphql.org/learn/.
10. *Apollo* [online] [cit. 2020-03-10]. Dostupné z: https://www.apollographql.com/platform.
11. *React Spring* [online] [cit. 2020-03-10]. Dostupné z: https://www.react-spring.io/.
12. *I18next* [online] [cit. 2020-03-10]. Dostupné z: https://www.i18next.com/overview/introduction.
13. *Právní Prostor* [online] [cit. 2020-03-11]. Dostupné z: httpshttp://www.pravniprostor.cz/.
14. *Hlídač ochranných známek* [online] [cit. 2020-05-03]. Dostupné z: https://atlasconsulting.cz/software/hlidac-ochrannych-znamek/.



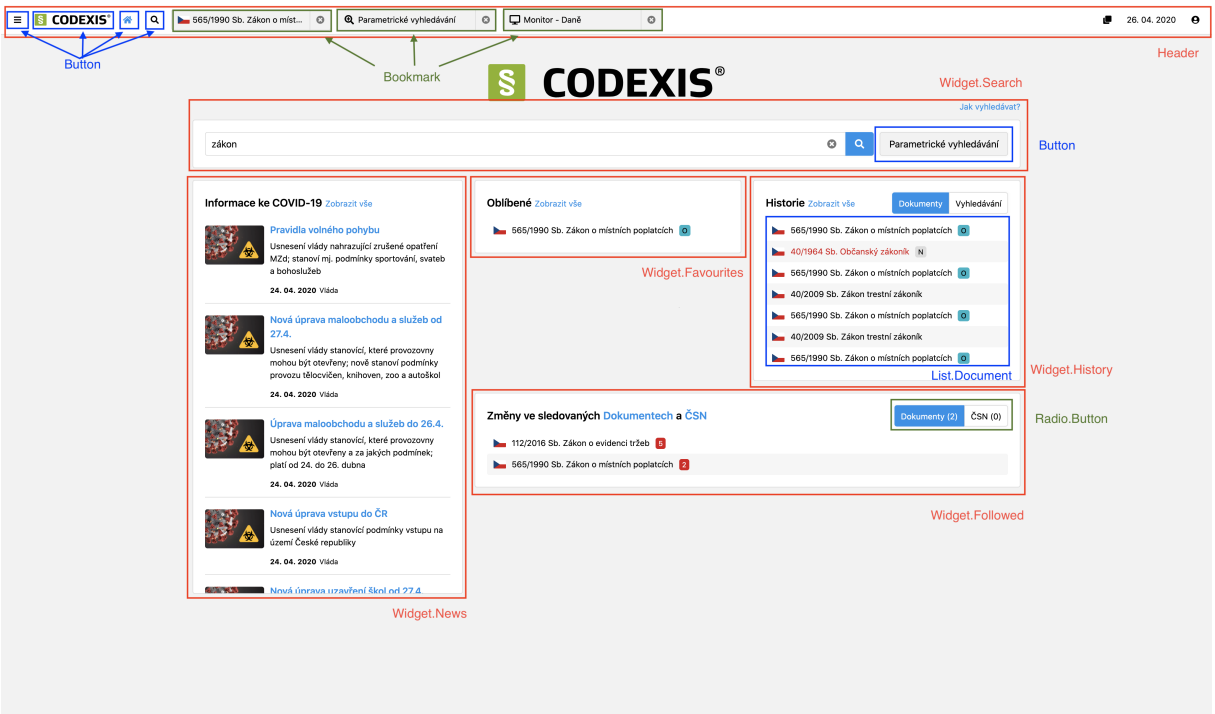
Obrázek 10: Storybook



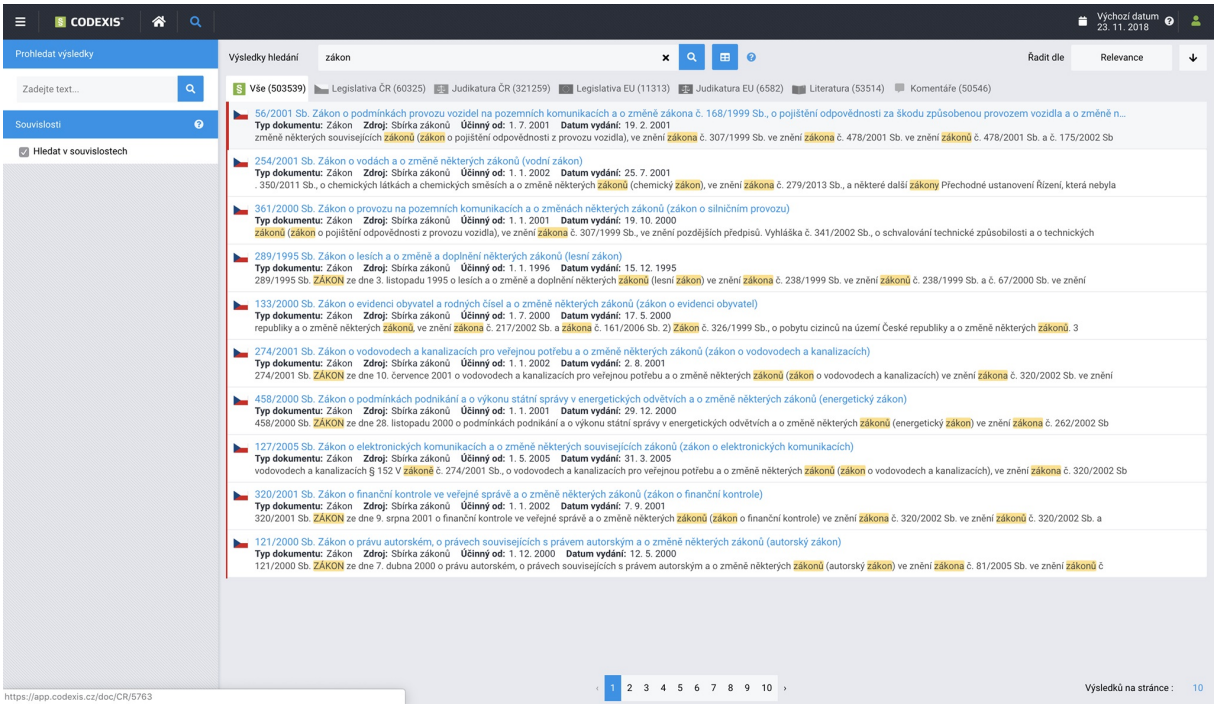
Obrázek 11: Titulní strana (původní)



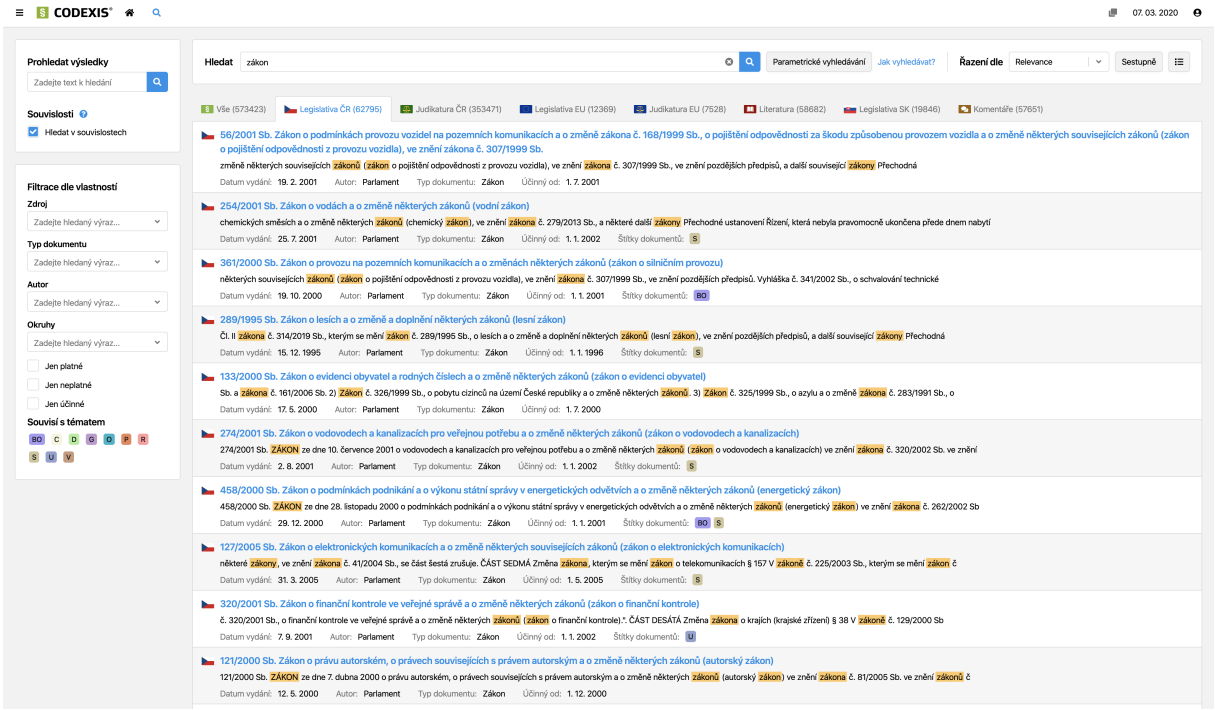
Obrázek 12: Titulní strana (nová)



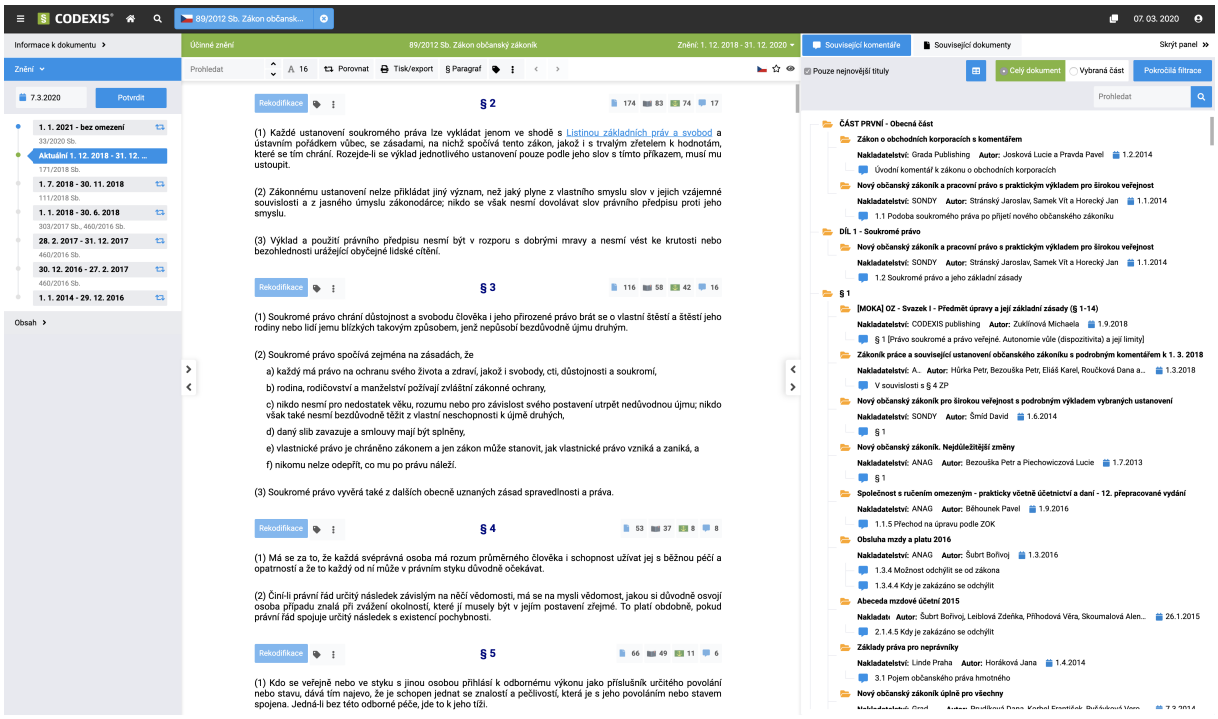
Obrázek 13: Titulní strana (rozložení komponent)



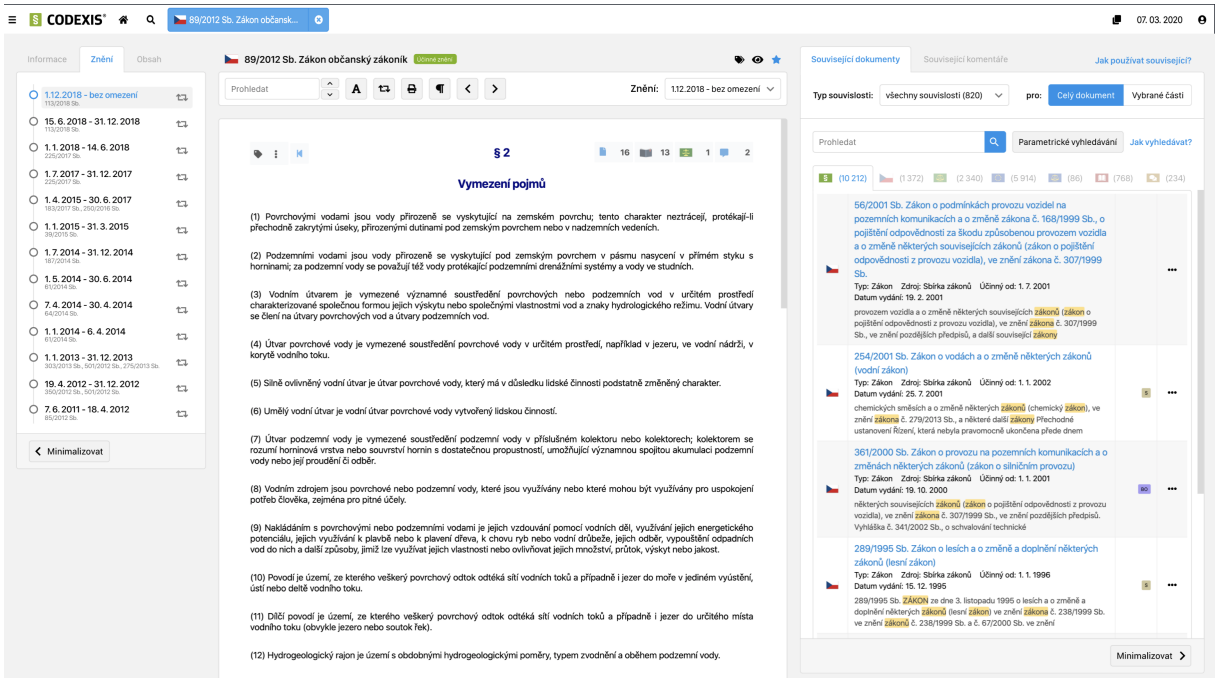
Obrázek 14: Vyhledávání (původní)



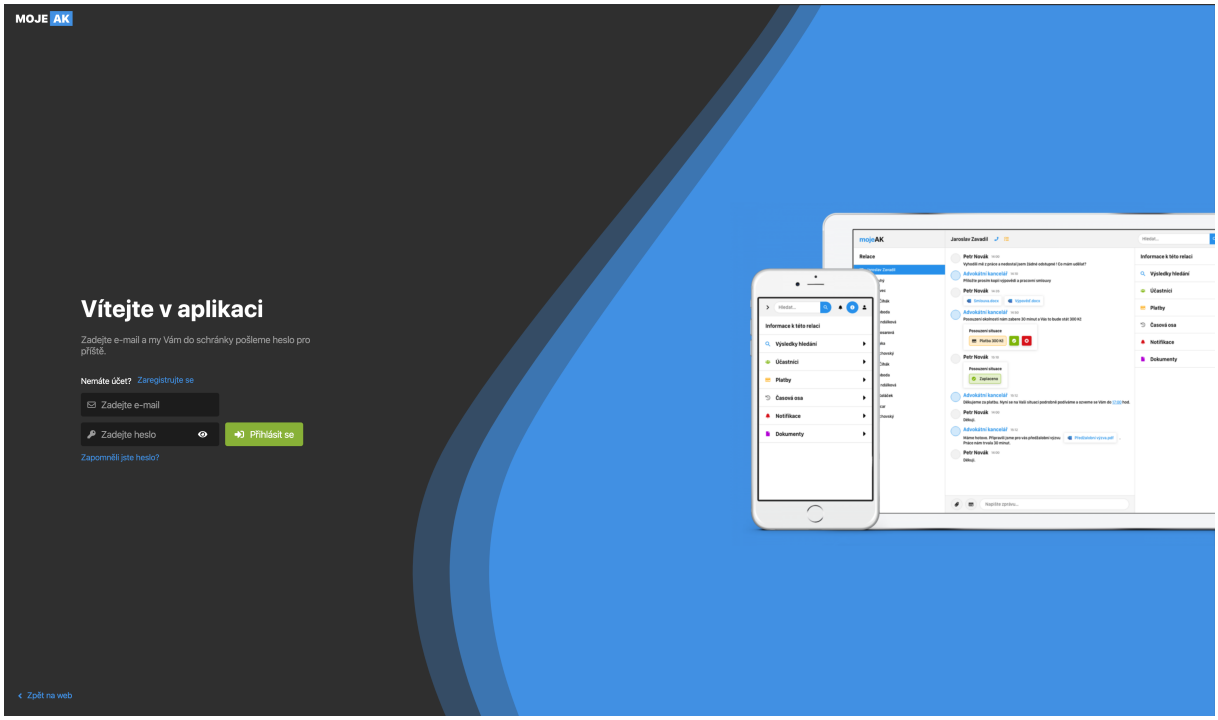
Obrázek 15: Vyhledávání (nové)



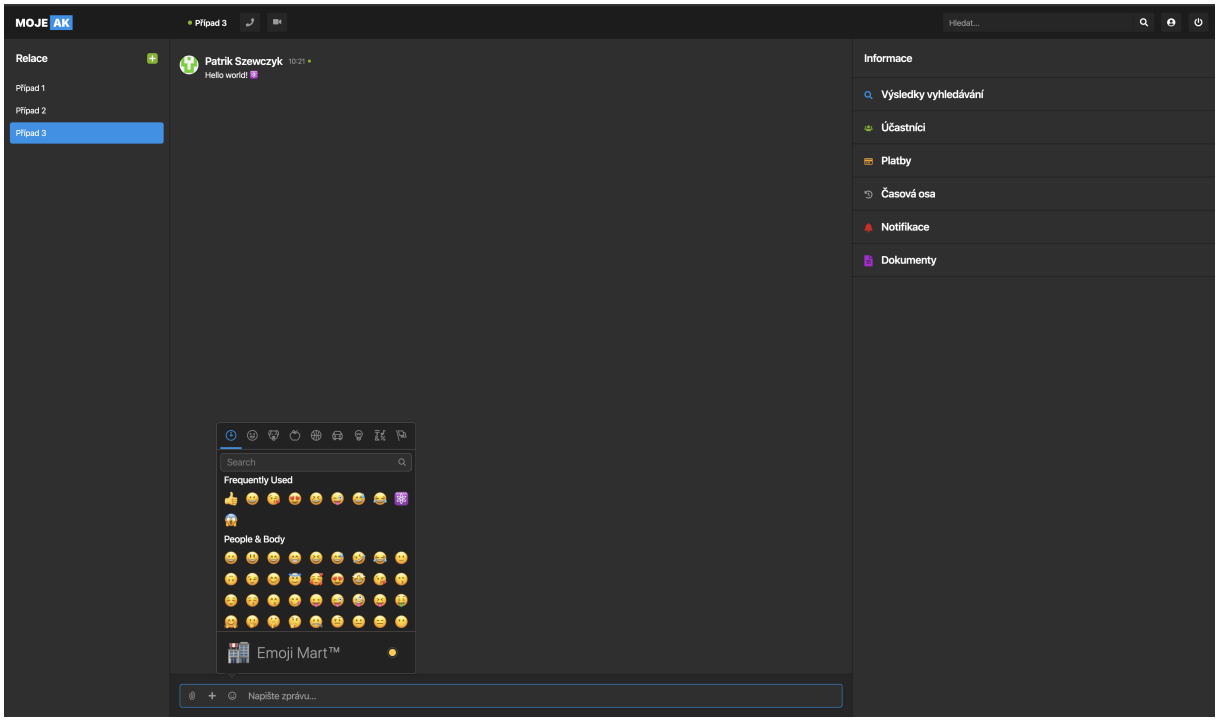
Obrázek 16: Dokument (původní)



Obrázek 17: Dokument (nový)



Obrázek 18: Project X (úvod)



Obrázek 19: Project X